

WHAT IS CLAIMED IS:

- 1 1. A computer-implemented method for managing a plurality
2 of processors as a virtual devices, said method
3 comprising:
4 receiving a device request at a task queue manager
5 running on a first processor in a computer system,
6 wherein the computer system includes a plurality of
7 heterogeneous processors that share a common memory
8 and wherein the device request corresponds to a
9 virtual device;
10 storing data corresponding to the request in the
11 common memory;
12 identifying a second processor from the plurality of
13 processors to handle the request;
14 signaling, from the first processor, the identified
15 second processor;
16 receiving the data corresponding to the request at the
17 second processor; and
18 processing the data by the second processor using
19 software code stored in the second processor's local
20 memory.
- 1 2. The method as described in claim 1 wherein the
2 identifying further comprises:
3 detecting a utilization of one or more processors from
4 the plurality of processors; and
5 selecting the second processor based upon its low
6 utilization.

- 1 3. The method as described in claim 1 wherein the
2 identifying further comprises:
3 detecting that the software code is loaded in the
4 second processor's local memory.
- 1 4. The method as described in claim 1 further comprising:
2 creating, by the first processor, a task block in the
3 common memory, the task block including a software
4 code identifier; and
5 writing the address of the task block to a mailbox
6 corresponding to the second processor.
- 1 5. The method as described in claim 4 further comprising:
2 receiving, at the second processor, the address of the
3 task block from the second processor's mailbox;
4 retrieving, at the second processor, the software code
5 identifier from the task block;
6 determining whether the software code corresponding to
7 the software code identifier is loaded in the second
8 processor's local memory; and
9 in response to determining that the software code
10 corresponding to the software code identifier is not
11 loaded in the second processor's local memory:
12 reading the software code from the common memory
13 into the second processor's local memory, wherein
14 the reading is performed using a DMA operation.
- 1 6. The method as described in claim 1 further comprising:

2 creating a plurality of queues, each of the queues
3 corresponding to a virtual device;
4 writing the device request to a first queue from the
5 plurality of queues, wherein the first queue
6 corresponds to the virtual device; and
7 assigning the first queue to the second processor.

1 7. The method as described in claim 1 further comprising:
2 creating, by the first processor, a task block in the
3 common memory, the task block including a software
4 code identifier and an input buffer address; and
5 writing the address of the task block to a mailbox
6 corresponding to the second processor.

1 8. The method as described in claim 7 further comprising:
2 receiving, at the second processor, the address of the
3 task block from the second processor's mailbox;
4 retrieving, at the second processor, the software code
5 identifier from the task block;
6 reading data from an input buffer located in the
7 common memory at a location corresponding to the input
8 buffer address into the second processor's local
9 memory, wherein the reading is performed using a DMA
10 operation;
11 determining whether the software code corresponding to
12 the software code identifier is loaded in the second
13 processor's local memory; and

14 in response to determining that the software code
15 corresponding to the software code identifier is not
16 loaded in the second processor's local memory:

17 reading the software code from the common memory
18 into the second processor's local memory, wherein
19 the reading is performed using a DMA operation.

1 9. The method as described in claim 1 further comprising:
2 receiving, at the first processor, a notification that
3 the second processor has completed the processing; and
4 notifying, by the first processor, a software process
5 from which the device request was received in response
6 to receiving the notification from the second
7 processor.

1 10. The method as described in claim 1 wherein the first
2 processor and the second processor are dislike
3 processors, wherein the first processor is adapted to
4 execute a first instruction set and wherein the second
5 processor is adapted to execute a second instruction
6 set.

1 11. A computer-implemented method for using a processor as
2 a virtual device, said method comprising:
3 initializing a task queue in a common memory, the task
4 queue corresponding to a device;
5 receiving a device request from a first processor of a
6 plurality processors in a computer system, wherein the
7 plurality of processors share the common memory;

8 storing the request in the task queue and storing the
9 input data corresponding to the request in the common
10 memory;

11 assigning the task queue to a second processor
12 selected from the plurality of processors;

13 signaling the second processor regarding the
14 assignment;

15 loading the input data into a local memory
16 corresponding to the second processor; and

17 processing the loaded input data using software code
18 executing on the second processor.

1 12. The method as described in claim 11 wherein the
2 assigning further comprises:

3 detecting a utilization of one or more processors from
4 the plurality of processors; and

5 selecting the second processor based upon its low
6 utilization.

1 13. The method as described in claim 11 wherein the
2 assigning further comprises:

3 detecting that the software code is loaded in the
4 second processor's local memory.

1 14. The method as described in claim 11 wherein the first
2 processor and the second processor are dislike
3 processors, wherein the first processor is adapted to
4 execute a first instruction set and wherein the second

5 processor is adapted to execute a second instruction
6 set.

1 15. The method as described in claim 11 further
2 comprising:

3 creating, by the first processor, a task block in the
4 common memory, the task block including a software
5 code identifier; and

6 writing the address of the task block to a mailbox
7 corresponding to the second processor.

1 16. The method as described in claim 15 further
2 comprising:

3 receiving, at the second processor, the address of the
4 task block from the second processor's mailbox;

5 retrieving, at the second processor, the software code
6 identifier from the task block;

7 determining whether the software code corresponding to
8 the software code identifier is loaded in the second
9 processor's local memory; and

10 in response to determining that the software code
11 corresponding to the software code identifier is not
12 loaded in the second processor's local memory:

13 reading the software code from the common memory
14 into the second processor's local memory, wherein
15 the reading is performed using a DMA operation.

1 17. The method as described in claim 11 further
2 comprising:

3 creating, by the first processor, a task block in the
4 common memory, the task block including a software
5 code identifier and the address of the input data; and
6 writing the address of the task block to a mailbox
7 corresponding to the second processor.

1 18. The method as described in claim 17 further
2 comprising:

3 receiving, at the second processor, the address of the
4 task block from the second processor's mailbox; and
5 retrieving, at the second processor, the software code
6 identifier and the address of the input data from the
7 task block, wherein the loading includes reading the
8 input data located at the input data address in the
9 common memory into the second processor's local memory
10 using a DMA operation.

1 19. The method as described in claim 11 further
2 comprising:

3 receiving, at the first processor, a notification that
4 the second processor has completed the processing; and
5 notifying, by the first processor, a software process
6 from which the device request was received in response
7 to receiving the notification from the second
8 processor.

1 20. The method as described in claim 19 wherein the
2 notifying further comprises:

3 writing completion data resulting from the processing
4 to a data structure; and
5 providing the data structure to software process.

1 21. An information handling system comprising:
2 a plurality of heterogeneous processors;
3 a common memory shared by the plurality of
4 heterogeneous processors;
5 a first processor selected from the plurality of
6 processors that sends a request to a second processor,
7 the second processor also being selected from the
8 plurality of processors;
9 a local memory corresponding to the second processor;
10 a DMA controller associated with the second processor,
11 the DMA controller adapted to transfer data between
12 the common memory and the second processor's local
13 memory; and
14 a virtual device tool for managing a plurality of
15 processors as a virtual devices, the virtual device
16 tool including software effective to:
17 receive a virtual device request at a task queue
18 manager running on the first processor;
19 store data corresponding to the request in the
20 common memory;
21 identify the second processor to handle the
22 request from the plurality of processors;

23 signal, from the first processor, the identified
24 second processor;

25 receive the data corresponding to the request at
26 the second processor; and

27 process the data by the second processor using
28 software code stored in the second processor's
29 local memory.

1 22. The information handling system as described in claim
2 21 wherein the identification of the second processor
3 further comprises software effective to:

4 detect a utilization of one or more processors from
5 the plurality of processors; and

6 select the second processor based upon its low
7 utilization.

1 23. The information handling system as described in claim
2 21 wherein the identification of the second processor
3 further comprises software effective to:

4 detect that the software code is loaded in the second
5 processor's local memory.

1 24. The information handling system as described in claim
2 21 further comprising software effective to:

3 create, by the first processor, a task block in the
4 common memory, the task block including a software
5 code identifier; and

6 write the address of the task block to a mailbox
7 corresponding to the second processor.

1 25. The information handling system as described in claim
2 24 further comprising software effective to:

3 receive, at the second processor, the address of the
4 task block from the second processor's mailbox;

5 retrieve, at the second processor, the software code
6 identifier from the task block;

7 determine whether the software code corresponding to
8 the software code identifier is loaded in the second
9 processor's local memory; and

10 in response to determining that the software code
11 corresponding to the software code identifier is not
12 loaded in the second processor's local memory,
13 executing software effective to:

14 read the software code from the common memory
15 into the second processor's local memory, wherein
16 the reading is performed using a DMA operation.

1 26. The information handling system as described in claim
2 21 further comprising software effective to:

3 create a plurality of queues, each of the queues
4 corresponding to a virtual device;

5 write the device request to a first queue from the
6 plurality of queues, wherein the first queue
7 corresponds to the virtual device; and

8 assign the first queue to the second processor.

1 27. The information handling system as described in claim
2 21 further comprising software effective to:

3 create, by the first processor, a task block in the
4 common memory, the task block including a software
5 code identifier and an input buffer address; and
6 write the address of the task block to a mailbox
7 corresponding to the second processor.

1 28. The information handling system as described in claim
2 27 further comprising software effective to:

3 receive, at the second processor, the address of the
4 task block from the second processor's mailbox;

5 retrieve, at the second processor, the software code
6 identifier from the task block;

7 read data from an input buffer located in the common
8 memory at a location corresponding to the input buffer
9 address into the second processor's local memory,
10 wherein the reading is performed using a DMA
11 operation;

12 determine whether the software code corresponding to
13 the software code identifier is loaded in the second
14 processor's local memory; and

15 in response to determining that the software code
16 corresponding to the software code identifier is not
17 loaded in the second processor's local memory, perform
18 software effective to:

19 read the software code from the common memory
20 into the second processor's local memory, wherein
21 the reading is performed using a DMA operation.

1 29. The information handling system as described in claim
2 21 further comprising software effective to:
3 receive, at the first processor, a notification that
4 the second processor has completed the processing; and
5 notify, by the first processor, a software process
6 from which the device request was received in response
7 to receiving the notification from the second
8 processor.

1 30. The information handling system as described in claim
2 21 wherein the first processor and the second
3 processor are dislike processors, wherein the first
4 processor is adapted to execute a first instruction
5 set and wherein the second processor is adapted to
6 execute a second instruction set.

1 31. An information handling system comprising:
2 a plurality of heterogeneous processors;
3 a common memory shared by the plurality of
4 heterogeneous processors;
5 a first processor selected from the plurality of
6 processors that sends a request to a second processor,
7 the second processor also being selected from the
8 plurality of processors;
9 a local memory corresponding to the second processor;
10 a DMA controller associated with the second processor,
11 the DMA controller adapted to transfer data between

12 the common memory and the second processor's local
13 memory; and

14 a virtual device tool for managing a plurality of
15 processors as a virtual devices, the virtual device
16 tool including software effective to:

17 initialize a task queue in the common memory, the
18 task queue corresponding to a virtual device;

19 receive a virtual device request from the first
20 processor;

21 store the request in the task queue and store the
22 input data corresponding to the request in the
23 common memory;

24 assign the task queue to the second processor
25 selected from the plurality of processors;

26 signal the second processor regarding the
27 assignment;

28 load the input data into a local memory
29 corresponding to the second processor; and

30 process the loaded input data using software code
31 executing on the second processor.

1 32. The information handling system as described in claim
2 31 wherein the assignment further comprises software
3 effective to:

4 detect a utilization of one or more processors from
5 the plurality of processors; and

6 select the second processor based upon its low
7 utilization.

1 33. The information handling system as described in claim
2 31 wherein the assignment further comprises software
3 effective to:

4 detect that the software code is loaded in the second
5 processor's local memory.

1 34. The information handling system as described in claim
2 31 wherein the first processor and the second
3 processor are dislike processors, wherein the first
4 processor is adapted to execute a first instruction
5 set and wherein the second processor is adapted to
6 execute a second instruction set.

1 35. The information handling system as described in claim
2 31 further comprising software effective to:

3 create, by the first processor, a task block in the
4 common memory, the task block including a software
5 code identifier; and

6 write the address of the task block to a mailbox
7 corresponding to the second processor.

1 36. The information handling system as described in claim
2 35 further comprising software effective to:

3 receive, at the second processor, the address of the
4 task block from the second processor's mailbox;

5 retrieve, at the second processor, the software code
6 identifier from the task block;

7 determine whether the software code corresponding to
8 the software code identifier is loaded in the second
9 processor's local memory; and

10 in response to determining that the software code
11 corresponding to the software code identifier is not
12 loaded in the second processor's local memory, perform
13 software effective to:

14 read the software code from the common memory
15 into the second processor's local memory, wherein
16 the reading is performed using a DMA operation.

1 37. The information handling system as described in claim
2 31 further comprising software effective to:

3 create, by the first processor, a task block in the
4 common memory, the task block including a software
5 code identifier and the address of the input data; and
6 write the address of the task block to a mailbox
7 corresponding to the second processor.

1 38. The information handling system as described in claim
2 37 further comprising software effective to:

3 receive, at the second processor, the address of the
4 task block from the second processor's mailbox; and
5 retrieve, at the second processor, the software code
6 identifier and the address of the input data from the
7 task block, wherein the loading includes reading the
8 input data located at the input data address in the
9 common memory into the second processor's local memory
10 using a DMA operation.

1 39. The information handling system as described in claim
2 31 further comprising software effective to:
3 receive, at the first processor, a notification that
4 the second processor has completed the processing; and
5 notify, by the first processor, a software process
6 from which the device request was received in response
7 to receiving the notification from the second
8 processor.

1 40. The information handling system as described in claim
2 39 wherein the notification of the software process
3 further comprises software effective to:
4 write completion data resulting from the processing to
5 a data structure; and
6 provide the data structure to software process.

1 41. A computer program product stored on a computer
2 operable media for managing a plurality of processors
3 as a virtual devices, said computer program product
4 comprising:
5 means for receiving a device request at a task queue
6 manager running on a first processor in a computer
7 system, wherein the computer system includes a
8 plurality of heterogeneous processors that share a
9 common memory and wherein the device request
10 corresponds to a virtual device;
11 means for storing data corresponding to the request in
12 the common memory;

13 means for identifying a second processor from the
14 plurality of processors to handle the request;
15 means for signaling, from the first processor, the
16 identified second processor;
17 means for receiving the data corresponding to the
18 request at the second processor; and
19 means for processing the data by the second processor
20 using software code stored in the second processor's
21 local memory.

1 42. The computer program product as described in claim 41
2 wherein the means for identifying further comprises:
3 means for detecting a utilization of one or more
4 processors from the plurality of processors; and
5 means for selecting the second processor based upon
6 its low utilization.

1 43. The computer program product as described in claim 41
2 wherein the means for identifying further comprises:
3 means for detecting that the software code is loaded
4 in the second processor's local memory.

1 44. The computer program product as described in claim 41
2 further comprising:
3 means for creating, by the first processor, a task
4 block in the common memory, the task block including a
5 software code identifier; and
6 means for writing the address of the task block to a
7 mailbox corresponding to the second processor.

1 45. The computer program product as described in claim 44
2 further comprising:

3 means for receiving, at the second processor, the
4 address of the task block from the second processor's
5 mailbox;

6 means for retrieving, at the second processor, the
7 software code identifier from the task block;

8 means for determining whether the software code
9 corresponding to the software code identifier is
10 loaded in the second processor's local memory; and

11 in response to determining that the software code
12 corresponding to the software code identifier is not
13 loaded in the second processor's local memory:

14 means for reading the software code from the
15 common memory into the second processor's local
16 memory, wherein the reading is performed using a
17 DMA operation.

1 46. The computer program product as described in claim 41
2 further comprising:

3 means for creating a plurality of queues, each of the
4 queues corresponding to a virtual device;

5 means for writing the device request to a first queue
6 from the plurality of queues, wherein the first queue
7 corresponds to the virtual device; and

8 means for assigning the first queue to the second
9 processor.

1 47. The computer program product as described in claim 41
2 further comprising:

3 means for creating, by the first processor, a task
4 block in the common memory, the task block including a
5 software code identifier and an input buffer address;
6 and

7 means for writing the address of the task block to a
8 mailbox corresponding to the second processor.

1 48. The computer program product as described in claim 47
2 further comprising:

3 means for receiving, at the second processor, the
4 address of the task block from the second processor's
5 mailbox;

6 means for retrieving, at the second processor, the
7 software code identifier from the task block;

8 means for reading data from an input buffer located in
9 the common memory at a location corresponding to the
10 input buffer address into the second processor's local
11 memory, wherein the reading is performed using a DMA
12 operation;

13 means for determining whether the software code
14 corresponding to the software code identifier is
15 loaded in the second processor's local memory; and

16 in response to determining that the software code
17 corresponding to the software code identifier is not
18 loaded in the second processor's local memory:

19 means for reading the software code from the
20 common memory into the second processor's local
21 memory, wherein the reading is performed using a
22 DMA operation.

1 49. The computer program product as described in claim 41
2 further comprising:

3 means for receiving, at the first processor, a
4 notification that the second processor has completed
5 the processing; and

6 means for notifying, by the first processor, a
7 software process from which the device request was
8 received in response to receiving the notification
9 from the second processor.

1 50. The computer program product as described in claim 41
2 wherein the first processor and the second processor
3 are dislike processors, wherein the first processor is
4 adapted to execute a first instruction set and wherein
5 the second processor is adapted to execute a second
6 instruction set.

1 51. A computer program product stored on a computer
2 operable media for managing a plurality of processors
3 as a virtual devices, said computer program product
4 comprising:

5 means for initializing a task queue in a common
6 memory, the task queue corresponding to a device;

7 means for receiving a device request from a first
8 processor of a plurality processors in a computer

9 system, wherein the plurality of processors share the
10 common memory;

11 means for storing the request in the task queue and
12 storing the input data corresponding to the request in
13 the common memory;

14 means for assigning the task queue to a second
15 processor selected from the plurality of processors;

16 means for signaling the second processor regarding the
17 assignment;

18 means for loading the input data into a local memory
19 corresponding to the second processor; and

20 means for processing the loaded input data using
21 software code executing on the second processor.

1 52. The computer program product as described in claim 51
2 wherein the means for assigning further comprises:

3 means for detecting a utilization of one or more
4 processors from the plurality of processors; and

5 means for selecting the second processor based upon
6 its low utilization.

1 53. The computer program product as described in claim 51
2 wherein the means for assigning further comprises:

3 means for detecting that the software code is loaded
4 in the second processor's local memory.

1 54. The computer program product as described in claim 51
2 wherein the first processor and the second processor
3 are dislike processors, wherein the first processor is

4 adapted to execute a first instruction set and wherein
5 the second processor is adapted to execute a second
6 instruction set.

1 55. The computer program product as described in claim 51
2 further comprising:

3 means for creating, by the first processor, a task
4 block in the common memory, the task block including a
5 software code identifier; and

6 means for writing the address of the task block to a
7 mailbox corresponding to the second processor.

1 56. The computer program product as described in claim 55
2 further comprising:

3 means for receiving, at the second processor, the
4 address of the task block from the second processor's
5 mailbox;

6 means for retrieving, at the second processor, the
7 software code identifier from the task block;

8 means for determining whether the software code
9 corresponding to the software code identifier is
10 loaded in the second processor's local memory; and
11 in response to determining that the software code
12 corresponding to the software code identifier is not
13 loaded in the second processor's local memory:

14 means for reading the software code from the
15 common memory into the second processor's local
16 memory, wherein the reading is performed using a
17 DMA operation.

1 57. The computer program product as described in claim 51
2 further comprising:

3 means for creating, by the first processor, a task
4 block in the common memory, the task block including a
5 software code identifier and the address of the input
6 data; and

7 means for writing the address of the task block to a
8 mailbox corresponding to the second processor.

1 58. The computer program product as described in claim 57
2 further comprising:

3 means for receiving, at the second processor, the
4 address of the task block from the second processor's
5 mailbox; and

6 means for retrieving, at the second processor, the
7 software code identifier and the address of the input
8 data from the task block, wherein the loading includes
9 reading the input data located at the input data
10 address in the common memory into the second
11 processor's local memory using a DMA operation.

1 59. The computer program product as described in claim 51
2 further comprising:

3 means for receiving, at the first processor, a
4 notification that the second processor has completed
5 the processing; and

6 means for notifying, by the first processor, a
7 software process from which the device request was

8 received in response to receiving the notification
9 from the second processor.

1 60. The computer program product as described in claim 59
2 wherein the means for notifying further comprises:
3 means for writing completion data resulting from the
4 processing to a data structure; and
5 means for providing the data structure to software
6 process.